IN THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Currently Amended) A method on an information processing system for automatic replacement of object classes, comprising:

performing static analysis on a program containing a plurality of objects in order to determine constraints on the transformations that can be applied and to detect unused functionality in one or more of the objects to be replaced;

analyzing the plurality of objects to detect usage patterns of functionality in the one or more objects replaced;

analyzing at least one execution of the program to collect profile information for the one or more objects; and

generating customized classes based upon the static analysis and the usage patterns detected and the profile information which has been collected.

2. (Original) The method according to claim 1, wherein the performing static analysis on a program containing a plurality of objects in order to determine constraints includes determining constraints which are type constraints.

3. (Original) The method according to claim 1, wherein the plurality of objects is a plurality of container objects.

4. (Original) The method according to claim 1, wherein the analyzing the plurality of objects includes instrumenting the plurality of objects to detect usage patterns of functionality in the one or more objects replaced.

5. (Original) The method according to claim 1, further comprising rewriting bytecode of an application to use the generated classes while providing transparency in the program's observable behavior during the replacement of the objects.

6. (Original) The method according to claim 1, wherein the performing static analysis further comprises performing static analysis to determine constraints by determining if the type of one or more objects to be replaced is a supertype of a type referenced in a cast expression.

7. (Original) The method according to claim 1, wherein the performing static analysis further comprises performing static analysis to determine type-correctness constraints by determining if the type of one or more objects to be replaced is a supertype of a type referenced in a cast expression.

8. (Original) The method according to claim 1, wherein the performing static analysis further comprises performing static analysis to determine interface-compatibility constraints in one or more of the objects to be replaced.

9. (Original) The method according to claim 1, wherein the performing static analysis further comprises performing static analysis to preserve run-time behavior for casts and instanceof operations for one or more of the objects to be replaced.

10. (Currently Amended) The method according to claim 1, wherein the performing static analysis includes using points-to ~~sets~~ analysis to determine where references to classes in allocation sites, declarations, casts and instanceof-expressions are modifiable to refer to one or more of the objects to be replaced.

11. (Currently Amended) The method according to claim 1, wherein the performing static analysis includes using points-to ~~sets~~ analysis to determine where references to container classes in allocation sites, declarations, casts and instanceof-expressions are modifiable to refer to one or more of the objects to be replaced.

12. (Original) The method according to claim 1, wherein the generating customized classes does not require a programmer to supply any additional types and additional external declarations for the customized classes.

13. (Currently Amended) The method according to claim 1, where the generating customized classes based upon the usage patterns detected includes:

identifying a customizable container class C with superclass B;

creating a class CustomC which contains methods and fields that are identical to those in class C ~~for each customizable container C with superclass B~~, wherein if B is not customizable, then CustomC's superclass is B, otherwise CustomC's superclass is CustomB;

introducing a type C^T ~~for each customizable container C~~, and making both C and CustomC ~~are made~~ a subtype of C^T and wherein type C^T contains declarations of all methods in C that are not declared in any superclass of C; and

introducing a type $C^\perp$ ~~is introduced for each customizable container C~~, and making $C^\perp$ ~~is made~~ a subclass of both C and CustomC , wherein type $C^\perp$ contains no methods, and wherein C^T and $C^\perp$ are intermediate types not provided as output during the generation of custom classes.

14. (Currently Amended) The method according to claim 13, wherein the generation of customized classes based upon the usage patterns detected includes:

determining at least one equivalence class~~es~~ E of declaration elements and expressions that must have the same type;

computing a set of possible types for ~~each of the~~ equivalence class~~es~~ E using an optimistic algorithm, wherein this algorithm associates a set S_E of types with ~~each~~ equivalence class E , ~~where class E is an instance of class C and~~ which is initialized as follows:

associating a set S_E with ~~an~~ the equivalence class ~~that~~ E containing the types C and CustomC if E contains an allocation site expression ~~E =~~ new C ~~, and initializing S_E with the types C and CustomC~~; and

associating a set S_E with ~~an~~ the equivalence class ~~that~~ E containing all types except ~~the~~ auxiliary types $C^{\top}$ and $C^{\perp}$, wherein $C^{\top}$ and $C^{\perp}$ are intermediate types not provided as output during the generation of custom classes if E does not contain any allocation site expressions~~, and initializing S_E with all types except the auxiliary types $C^{\top}$ and $C^{\perp}$, wherein $C^{\top}$ and $C^{\perp}$ are intermediate types not provided as output during the generation of custom classes~~.

15. (Currently Amended) The method according to claim 14, further comprising:

identifying sets S_D and S_E for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$;

removing from ~~a~~ set S_D ~~from~~ any type that is not a subtype of a type that occurs in S_E ~~for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$~~; and

removing from set S_E any type that is not a supertype of a type that occurs in S_D~~S_E is removed any type that is not a supertype of a type that occurs in S_E for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$.~~;

wherein the removing of ~~types from~~ S_D and S_E is performed repeatedly until a fixed point is reached.

16. (Currently Amended) A computer readable storage medium containing programming instructions for automatic replacement of object classes, the programming instructions comprising:

performing static analysis on a program containing a plurality of objects in order to determine constraints on the transformations that can be applied and to detect unused functionality in one or more of the objects to be replaced;

analyzing the plurality of objects to detect usage patterns of functionality in the one or more objects replaced;

analyzing at least one execution of the program to collect profile information for the one or more objects; and

generating customized classes based upon the static analysis and the usage patterns detected and the profile information which has been collected.

17. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis on a program containing a plurality of objects in order to determine constraints includes determining constraints which are type constraints.

18. (Currently Amended) The computer readable storage medium according to claim 16, wherein the plurality of objects is a plurality of container objects.

19. (Currently Amended) The computer readable storage medium according to claim 16, wherein the analyzing the plurality of objects includes instrumenting the plurality of objects to detect usage patterns of functionality in the one or more objects replaced.

20. (Currently Amended) The computer readable storage medium according to claim 16, further comprising rewriting bytecode of an application to use the generated classes while providing transparency in the program during the replacement of the objects.

21. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis further comprises performing static analysis to determine constraints by determining if a type of in one or more objects to be replaced is a supertype of a type referenced in a cast expression.

22. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis further comprises performing static analysis to determine type-correctness constraints by determining if a type of in one or more objects to be replaced is a supertype of a type referenced in a cast expression.

23. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis further comprises performing static analysis to determine interface-compatibility constraints in one or more of the objects to be replaced.

24. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis further comprises performing static analysis to preserve run-time behavior for casts and instanceof operations for one or more of the objects to be replaced.

25. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis includes using points-to~~-sets~~ analysis to determine where references to classes in allocation sites, declarations, casts and instanceof-expressions are modifiable to refer to one or more of the objects to be replaced.

26. (Currently Amended) The computer readable storage medium according to claim 16, wherein the performing static analysis includes using points-to ~~sets~~ analysis to determine where references to container classes in allocation sites, declarations, casts and instanceof-expressions are modifiable to refer to one or more of the objects to be replaced.

27. (Currently Amended) The computer readable storage medium according to claim 16, wherein the generating customized classes does not require a programmer to supply any additional types and additional external declarations for the customized classes.

28. (Currently Amended) The computer readable storage medium according to claim 27, wherein the generation customized classes based upon the usage patterns detected includes:

identifying a customizable container class C with superclass B;

creating a class CustomC which contains methods and fields that are identical to those in class C ~~for each customizable container C with superclass B~~, wherein if B is not customizable, then CustomC's superclass is B, otherwise CustomC's superclass is CustomB;

introducing a type C^T ~~for each customizable container C~~, and making both C and CustomC ~~are made~~ a subtype of C^T and wherein type C^T contains declarations of all methods in C that are not declared in any superclass of C; and

introducing a type $C^{\perp}$~~is introduced for each customizable container C~~, and making $C^{\perp}$ ~~is made~~ a subclass of both C and CustomC , wherein type $C^{\perp}$ contains no methods, and wherein C^T and $C^{\perp}$ are intermediate types not provided as output during the generation of custom classes.

29. (Currently Amended) The computer readable storage medium according to claim 28, wherein the generation of customized classes based upon the usage patterns detected includes:

determining at least one equivalence class~~es~~ E of declaration elements and expressions that must have the same type;

computing a set of possible types for ~~each of the~~ equivalence class~~es~~ E using an optimistic algorithm, wherein this algorithm associates a set S_E of types with ~~each~~ equivalence class E , ~~where class E is an instance of class C and~~ which is initialized as follows:

associating a set S_E with ~~an~~ the equivalence class ~~that~~ E containing the types C and CustomC if E contains an allocation site expression ~~E =~~ new C ~~, and initializing S_E with the types C and CustomC~~; and

associating a set S_E with ~~an~~ the equivalence class ~~that~~ E containing all types except ~~the~~ auxiliary types $C^T$and $C^\perp$, wherein C^T and $C^\perp$ are intermediate types not provided as output during the generation of custom classes if E does not contain any allocation site expressions~~, and initializing S_E with all types except the auxiliary types $C^T$and $C^\perp$, wherein $C^T$and $C^\perp$are intermediate types not provided as output during the generation of custom classes~~.

30. (Currently Amended) The computer readable storage medium according to claim 29, further comprising:

identifying sets S_D and S_E for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$;

removing from ~~a~~ set S_D ~~from~~ any type that is not a subtype of a type that occurs in S_E ~~for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$~~; and

removing from set S_E any type that is not a supertype of a type that occurs in S_D~~S_E is removed any type that is not a supertype of a type that occurs in S_E for each pair of equivalence classes D, E such that there exists a type constraint $D \leq E$~~;

wherein the removing of ~~types from~~ S_D and S_E is performed repeatedly until a fixed point is reached.

31. (Currently Amended) An information processing system with programming instructions for automatic replacement of object classes, comprising:

means for performing static analysis on a program containing a plurality of objects in order to determine constraints on the transformations that can be applied and to detect unused functionality in one or more of the objects to be replaced;

means for analyzing the plurality of objects to detect usage patterns of functionality in the one or more objects replaced;

means for analyzing at least one execution of the program to collect profile information for the one or more objects; and

means for generating customized classes based upon the static analysis and the usage patterns detected and the profile information which has been collected.